

User Manual
for
AD160-12-8M-104

**Single Channel, 60 MSPS, 12-bit,
PC/104 Analog Input Module**

CHASE SCIENTIFIC COMPANY

P.O. Box 1487
Langley, WA 98260

Tel: 360-221-8455
Fax: 360-221-8457

Email: techsupport@chase2000.com
Web: <http://www.chase2000.com>

Original Document: AD160-12-8M-104_manual.odt
(created 10-23-2006, updated 4/5/2007)

© Copyright 2006 by
Chase Scientific Company

This manual, the AD160 module, and the software drivers outlined in this document are copyrighted with all rights reserved. Under the copyright laws, the above mentioned may not be copied, in whole or in part, without the express written consent of Chase Scientific Company.

TABLE OF CONTENTS

1 GENERAL INFORMATION.....	4
1.1 INTRODUCTION.....	4
1.2 REFERENCES.....	4
1.3 DELIVERABLES.....	4
1.3.1 Software.....	4
1.3.2 Hardware.....	5
1.3.3 Checklist.....	5
1.4 PRODUCT SPECIFICATION.....	5
1.5 TECHNICAL SUPPORT / SOFTWARE UPDATES.....	6
1.6 WARRANTY.....	6
2 HARDWARE DESCRIPTION.....	7
2.1 INTRODUCTION.....	7
2.2 BLOCK DIAGRAM.....	7
2.3 BOARD DRAWING.....	8
2.4 BOARD I/O.....	9
2.4.1 Header Pinouts and Pin Descriptions.....	9
2.5 HARDWARE REGISTER DEFINITIONS.....	9
2.6 ANALOG INPUT RANGES AND RESOLUTION.....	9
3 THEORY OF OPERATION.....	10
3.1 INTRODUCTION.....	10
3.2 ACQUIRING ANALOG INPUT DATA.....	10
4 SOFTWARE DRIVERS.....	10
4.1 INTRODUCTION.....	10
4.2 DRIVER INSTALLATION.....	10
4.2.1 16-Bit DOS.....	10
4.2.2 Linux Installation (Kernels 2.2x, 2.4x, 2.6x).....	10
4.2.3 Windows 98/ME/2000/XP.....	11
4.3 FUNCTION CALLS.....	11
4.3.1 Function Declarations in C.....	11
4.3.2 Function Call Descriptions / Usage.....	13
4.3.2.1 ad160_SetBasePortAddr().....	13
4.3.2.2 ad160_Reset().....	13
4.3.2.3 ad160_SetClockRate().....	14
4.3.2.4 ad160_Set_Memory_Size().....	14
4.3.2.5 SetTrigPol().....	15
4.3.2.6 ad160_Digitize().....	15
4.3.2.7 ad160_DIG_DONE().....	16
4.3.2.8 ad160_Transfer_Data().....	16
4.3.2.9 ad160_PowerDown().....	17
4.4 PROGRAMMING EXAMPLES.....	17
4.4.1 DOS 16-BIT C Example.....	17
4.4.2 LINUX C/C++ Example.....	19
5 MISCELLANEOUS.....	21
5.1 CALIBRATION.....	21
5.2 MAINTENANCE.....	21

ILLUSTRATIONS / TABLES

FIGURE 1 – BLOCK DIAGRAM..... 8

FIGURE 2 – BOARD LAYOUT..... 9

FIGURE 3 - 4-PIN DIGITAL I/O CONNECTOR..... 10

1 GENERAL INFORMATION

1.1 Introduction

The AD160-12-8M-104 is a PC/104 analog I/O module which features one (1) 12-bit, high speed singled-ended (50 ohms) analog input channel capable of capturing data at up to 60 MSPS with a minimum bandwidth of 20 Mhz (100 Mhz optional). The AD160 module also comes standard with power down capability and operates over industrial temperature range standard (40°C to 85°C).

The analog input and trigger inputs are available on SMA connectors. The Analog input has up to 8Meg of software selectable onboard memory. The trigger source can be either an external TTL trigger or on-board software trigger.

Timing is controlled by an onboard clock circuit referenced to a 60 MHz crystal oscillator and can be programmed for 60MHz, 30MHz, 15MHz, and 7.5MHz. Once A/D conversions are completed, data can be transferred from the module to main memory via I/O port reads using simple function calls.

1.2 References

“P996.1 Standard for Compact Embedded-PC Modules”; PC/104 Specification, Version 2.3, June 1996 by PC/104 Consortium.

“IEEE P996 draft standard”, 1987 by IEEE Standards Committee. This bus standard was the basis for the first PC/104 standard released in 1992.

1.3 Deliverables

1.3.1 Software

The AD160 comes with source code drivers for **16-bit DOS and 32-bit Linux**. Software comes on a single 3.5” diskette or Mini-CD. Call Chase Scientific for the latest information on drivers for other operating system platforms or check the web site for the latest updates.

Software drivers are provided as source code in DOS 16-bit Borland C++ 5.2 code and Linux GNU GCC Version 4.1.0.

Actual Listing of files on Diskette/CD for Windows:

```
----- DIRECTORIES / FILES -----
AD160_Drivers (BASE_DIR)
|
|  readme.txt                // This file.
|  AD160-12-8M-104_manual.pdf // Manual
|
|  DOS_6p22                  // Directory for 16-Bit Borland C Drivers
|  |  AD160drv.c             // AD160 Driver Source
|  |  AD160DRV.H            // AD160 Driver Header
|  |  AD160_EX.C            // AD160 Example Source (calls driver)
|  |  AD160.EXE             // AD160 Compiled Example (uses Free Borland Turbo C 2.02)
|
|  LINUX_FC-5_2.6.15        // Directory for GNU GCC Version 4.1.0
|  |  ad160drv_o.cpp        // AD160 Driver Source
```

```

| | ad160drv_o.H           // AD160 Driver Header
| | ad160_test.cpp        // AD160 Example Source (calls driver)
| | ad160_test            // Example of Executable Compiled on Fedora Core 5.
| | makefile              // Type "makefile" to Compile Example code into Executable
"ad160_test".

----- E N D -----

```

1.3.2 Hardware

The AD160 hardware consists of a single PC/104 compliant module. The module is shipped with a manual with complete hardware and software descriptions. The ISA bus connector (PC/104 only portion) is the only connector on the card. There is no active or passive pass through PC/104-Plus connector.

1.3.3 Checklist

Item #	Qty	Part Number	Description
1	1	AD160-12-8M-104	Single Channel 60 MSPS, 12-bit, A/D PC/104 Module
2	1	AD160-Drivers	3.5" diskette with 16-Bit DOS source code and 32-Bit Linux source code. Includes examples.
3	1	AD160-Manual	User manual for AD160 board and software drivers.

1.4 Product Specification

(all specifications are at 25C unless otherwise specified)

HIGH SPEED ANALOG INPUTS (CH1-2)

Number of inputs	(1) Synchronous, Single Ended into 50 ohms
A/D Resolution	12-bit (1 part in 4096)
Input Bandwidth	20 MHz minimum @ 1V pk to pk
Input Range	The A/D count of 0 equals -0.525V. A count of 4095 equals +0.525V. (order option 3 for user specified FIXED gain)
Acquisition Time to Full-Scale Step	16 ns typical to 0.01% (1/2 LSB)
Input Coupling	50 ohms load to ground/shield
Overvoltage Protection	+/- 6V Maximum
Nonlinearity	+/- 2 LSB typical
Memory Size	8048576 samples maximum, 1048576 standard
Segment Sizes	Software selectable memory sizes of 128, 256, ..., 8048576 in multiples of 2.
Acquisition Mode	Single-shot initiated by software, external or internal triggering (see trigger sources)
Timebase	60 MHz, 30 MHz, 15 MHz, 7.5 Mhz (Software Selectable)
Trigger	
Location	Fixed trigger position at beginning of each segment
Sources	external TTL signal, internal software trigger
Slope	Positive, Negative
Coupling	DC

GENERAL

Power Supply (Vcc)	+5V @ 410mA (Ave. Running) +5V @ 130mA (PWR DWN)
Operating Temperature	-40 to +85 degrees C standard
Operating Humidity	5 to 95% non-condensing
Size	PC/104: 3.55" x 3.775"
Data Bus	16-bits PC/104 (i.e. ISA)

1.5 Technical Support / Software Updates

For technical support:

Phone	(360) 221-8455
Fax	(360) 221-8457
Email	techsupport@chase2000.com
Mail	Chase Scientific Company P.O. Box 1487 Langley, WA 98260

For software updates:

Email	techsupport@chase2000.com
Web	http://www.chase2000.com

1.6 Warranty

Chase Scientific Company (hereafter called Chase Scientific) warrants to the original purchaser that its AD160-12-1M-104P, and the component parts thereof, will be free from defects in workmanship and materials for a period of ONE YEAR from the data of purchase.

Chase Scientific will, without charge, repair or replace at its option, defective or component parts upon delivery to Chase Scientific's service department within the warranty period accompanied by proof of purchase date in the form of a sales receipt.

EXCLUSIONS: This warranty does not apply in the event of misuse or abuse of the product or as a result of unauthorized alterations or repairs. It is void if the serial number is altered, defaced or removed.

Chase Scientific shall not be liable for any consequential damages, including without limitation damages resulting from loss of use. Some states do not allow limitation or incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific rights. You may also have other rights that vary from state to state.

Chase Scientific warrants products sold only in the USA and Canada. In countries other than the USA, each distributor warrants the Chase Scientific products that it sells.

NOTICE: Chase Scientific reserves the right to make changes and/or improvements in the product(s) described in this manual at any time without notice.

2 HARDWARE DESCRIPTION

2.1 Introduction

The AD160 hardware consists of the following major functions:

- (1) 12-bit, 60 MSPS synchronous, single-ended, bipolar A/D Inputs (DC coupled)
- (1) TTL Trigger Input

2.2 Block Diagram

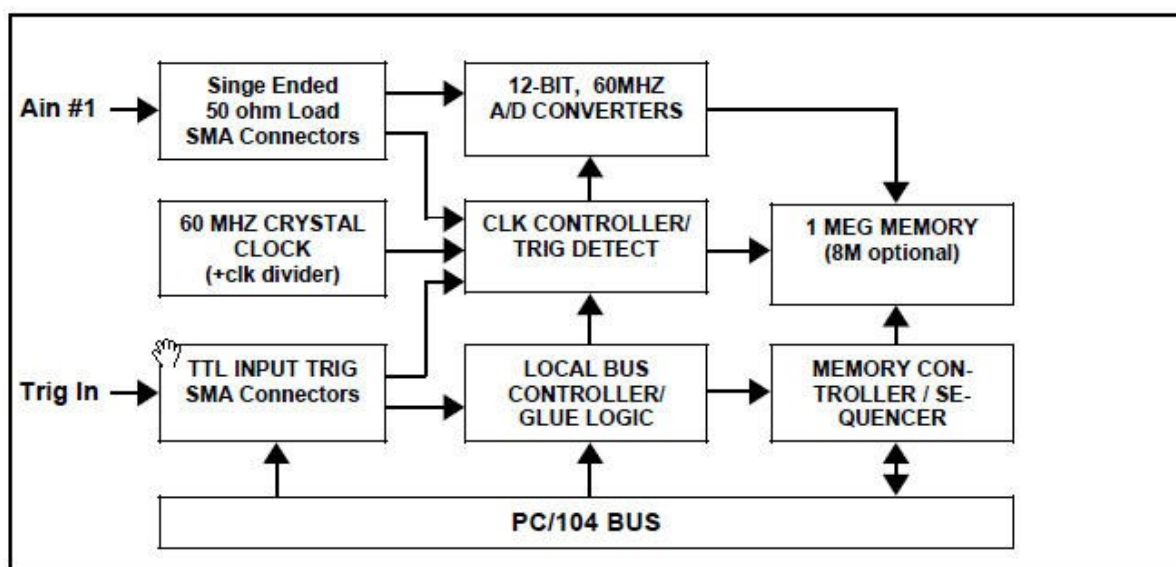


Figure 1 – Block Diagram

2.3 Board Drawing

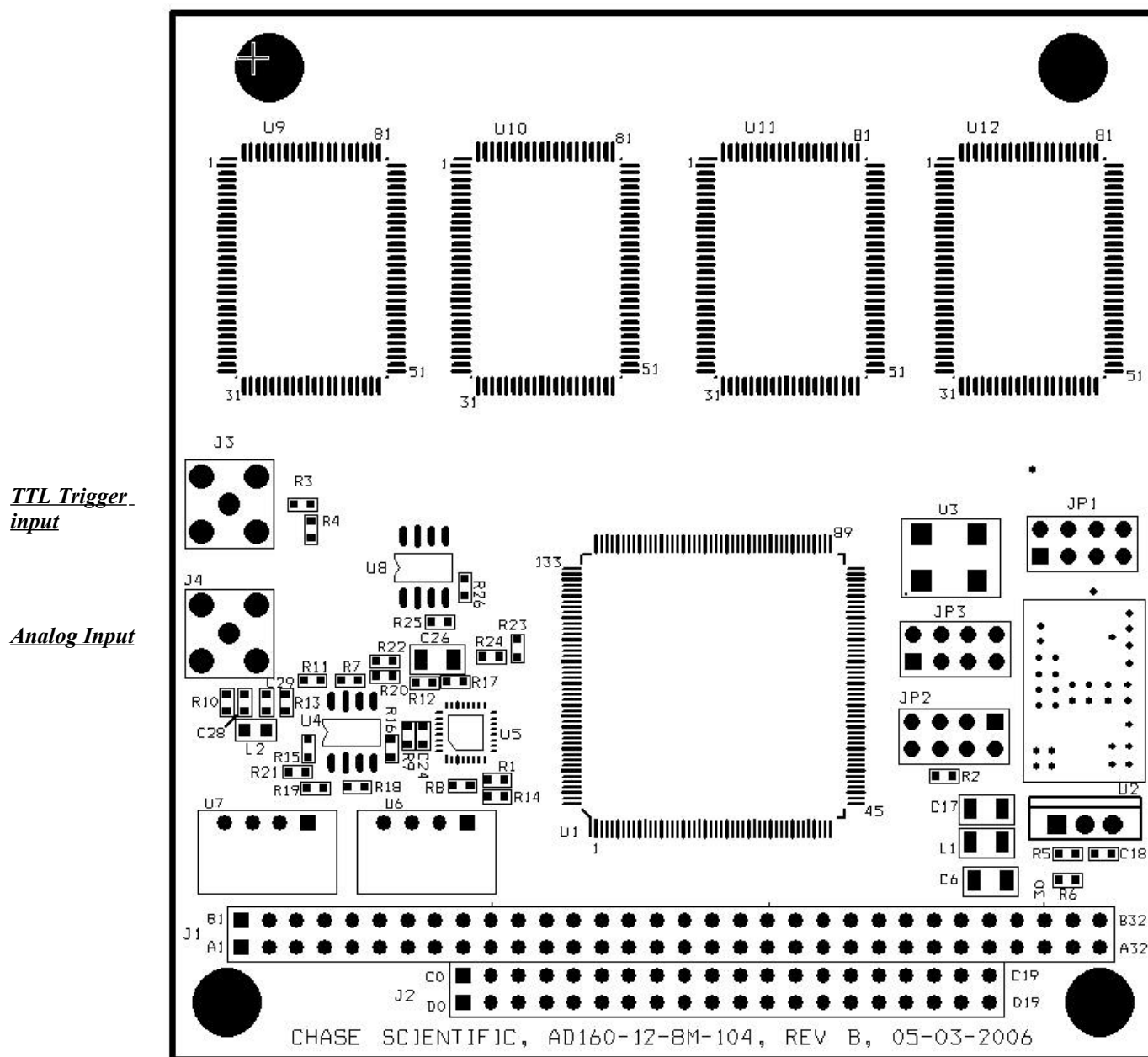


Figure 2 – Board Layout

2.4 Board I/O

2.4.1 Header Pinouts and Pin Descriptions

N/A

2.5 Hardware Register Definitions

Due to hardware protocol complexity, the details on how to use these ports are beyond the scope of this manual. The software drivers should provide an effortless integration path for the user. However, if the need arises that is beyond what the software drivers can provide, please call the factory for technical support.

2.6 Analog Input Ranges and Resolution

Analog input range is by default +/- 0.528V. Consult factory for user gain setting options. The resolution is 12 bits.

3 THEORY OF OPERATION

3.1 Introduction

Because the PC/104 bus is significantly slower than the digitizing rate of the AD160, the data must first be digitized and stored on the AD160 and then transferred to system memory at the slower bus speed.

3.2 Acquiring Analog Input Data

Before capturing data, the user must first setup the correct digitizing rate, determine if they're going to use external or internal trigger mode, and trigger polarity.

After proper setup is performed, you're ready to digitize. When the digitize function is called, the A/D converter is then activated after the first trigger event and the digitized analog signal is stored into buffer RAM. Data is then transferred to system RAM using another function call.

IMPORTANT:

The RAM on the AD160 is designed so that every time a sample is read out, an on-board address counter is automatically incremented. What this means is that if the user is limited in how much data they can read out at a given time - due to 16-bit DOS for example - then the user can read the samples out in 16K bursts and then store the data in another location such as a solid state disk drive or specially configured memory.

4 SOFTWARE DRIVERS

4.1 Introduction

The primary objective in designing software drivers is to get the user up and running as quickly as possible. The details on installing software drivers are listed in 4.2.xx. While the listing of function calls and their parameter definitions are listed in section 4.3.xx, the programming examples in section 4.4.x will show you how to include them into your programs.

The driver source codes are designed to work under all versions of 16-bit DOS and Linux with Kernel Code versions 2.2.x through 2.6.x.

4.2 Driver Installation

4.2.1 16-Bit DOS

DOS drivers are much simpler to use than Windows drivers, especially if they are not RAM resident types like they are in this case. Just include (i.e. `"#include \"ad160drv.c\""`) in your program and you're done. See sample code `"ad160_example.cpp"` for more information.

4.2.2 Linux Installation (Kernels 2.2x, 2.4x, 2.6x)

The Linux source driver code is almost as easy to use as the DOS driver code. We have provided a make file and example user code for your convenience so that you have a place to start. Please study the example file “ad160_test.cpp” thoroughly.

4.2.3 Windows 98/ME/2000/XP

Not applicable at this time.

4.3 Function Calls

4.3.1 Function Declarations in C

DOS HEADER FILE

```

/*
/* AD160 Header File for 16-Bit Borland Turbo C 2.02 */
/* ===== */
/*
/* Filename: ad160drv.h
/*
/*
/* Web site: http://www.chase2000.com
/* Email: support@chase2000.com
/*
/*
/* (c) Chase Scientific 2006
/*
/*

#ifndef ad160drvH
#define ad160drvH

typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned long DWORD;
typedef void *PVOID;

DWORD Memory_Size;

WORD BasePortAddr = 0x240; /* DEFAULT BASE PORT ADDRESS - Change as needed */

/***** USER ROUTINES *****/
void ad160_SetBasePortAddr(WORD PortAddr); /* AD160 default = 0x240 */
void ad160_Reset(void);
void ad160_SetClockRate(BYTE CLK_SEL); /* 1-4 ==> 60,30,15,7.5 MHz */
void ad160_Set_Memory_Size(BYTE MemSize); /* SEE BELOW FOR SIZES */
void SetTrigPol(BYTE TrigPol);
void ad160_Digitize(BYTE UseExtTrig);

BYTE ad160_DIG_DONE(void);
BYTE ad160_Transfer_Data(PVOID UserArray,DWORD NumSamples);
void ad160_PowerDown(BYTE PwrDwn);

/*
    if (MemSize == 4) Memory_Size = 16;
    if (MemSize == 5) Memory_Size = 32;
    if (MemSize == 6) Memory_Size = 64;
    if (MemSize == 7) Memory_Size = 128;

```

```

        if (MemSize == 8) Memory_Size = 256;
        if (MemSize == 9) Memory_Size = 512;
        if (MemSize == 10) Memory_Size = 1024;
        if (MemSize == 11) Memory_Size = 2048;
        if (MemSize == 12) Memory_Size = 4096;
        if (MemSize == 13) Memory_Size = 8192;
        if (MemSize == 14) Memory_Size = 16384;

        if (MemSize == 15) Memory_Size = 32768;
        if (MemSize == 16) Memory_Size = 65536;
        if (MemSize == 17) Memory_Size = 131072;
        if (MemSize == 18) Memory_Size = 262144;
        if (MemSize == 19) Memory_Size = 524288;
        if (MemSize == 20) Memory_Size = 1048576;
        if (MemSize == 21) Memory_Size = 2097152;
        if (MemSize == 22) Memory_Size = 4194304;
        if (MemSize == 23) Memory_Size = 8388608;
    */

#endif

```

LINUX HEADER FILE

```

//-----
// Filename: adl60drv_o.H
//-----
//
// 6-10-2006          Debug on Fedora 5 (Linux Version 2.6.15)
//                   - Downloaded 3/20/06
//
//-----
// Web site: http://www.chase2000.com
// Email:   support@chase2000.com
//
// (c) Chase Scientific 2006
//
//-----
//-----

#ifndef _adl60drv_O_H
#define _adl60drv_O_H

typedef unsigned char  BYTE;
typedef unsigned short WORD;
typedef unsigned long  DWORD;
typedef void *PVOID;

/***** USER ROUTINES *****/
void adl60_SetBasePortAddr(WORD PortAddr);    // AD160 default = 0x240
void adl60_Reset(void);
void adl60_SetClockRate(BYTE CLK_SEL);  // 0-3 ==> 60,30,15,7.5 MHz
void adl60_Set_Memory_Size(BYTE MemSize);  // SEE BELOW FOR SIZES
void SetTrigPol(BYTE TrigPol);
void adl60_Digitize(BYTE UseExtTrig);

BYTE adl60_DIG_DONE(void);
BYTE adl60_Transfer_Data(PVOID UserArray,DWORD NumSamples);
void adl60_PowerDown(BYTE PwrDwn);

/*

```

```

if (MemSize == 4) Memory_Size = 16;
if (MemSize == 5) Memory_Size = 32;
if (MemSize == 6) Memory_Size = 64;
if (MemSize == 7) Memory_Size = 128;
if (MemSize == 8) Memory_Size = 256;
if (MemSize == 9) Memory_Size = 512;
if (MemSize == 10) Memory_Size = 1024;
if (MemSize == 11) Memory_Size = 2048;
if (MemSize == 12) Memory_Size = 4096;
if (MemSize == 13) Memory_Size = 8192;
if (MemSize == 14) Memory_Size = 16384;
if (MemSize == 15) Memory_Size = 32768;
if (MemSize == 16) Memory_Size = 65536;
if (MemSize == 17) Memory_Size = 131072;
if (MemSize == 18) Memory_Size = 262144;
if (MemSize == 19) Memory_Size = 524288;
if (MemSize == 20) Memory_Size = 1048576;
if (MemSize == 21) Memory_Size = 2097152;
if (MemSize == 22) Memory_Size = 4194304;
if (MemSize == 23) Memory_Size = 8388608;
*/

#endif

```

4.3.2 Function Call Descriptions / Usage

4.3.2.1 ad160_SetBasePortAddr()

Description

Sets the base I/O port address which the remaining function calls use. The AD160 card has a default value of 0x240 as the base address. At the time of the this writing, the base address can only be programmed at the factory.

Declaration

```
void ad160_SetBasePortAddr(WORD PortAddr); // AD160 default = 0x240
```

Parameters

PortAddr: Hex Value 0x240 only.

Return Value

None.

Example

```
ad160_SetBasePortAddr(0x240);
```

4.3.2.2 ad160_Reset()

Description

Activates a master hardware reset function on the AD160. This primarily returns the card to a ready state by resetting the memory counters and triggering functions (re-arms). This should be called immediately after opening card for the first time and every time before "ad160_Digitize" is called. This function call does not effect Trigger Polarity, Memory Size, or Clock Rate previously set using other function calls.

Declaration

```
void ad160_Reset(void);
```

Parameters

None.

Return Value

None.

Example

```
ad160_Reset(); // Resets card before ad160_Digitize() is called.
```

4.3.2.3 ad160_SetClockRate()**Description**

Closes AD160 drivers. Should be called after finishing using the driver. However, if no other software uses the “windrv.xxx” (usual situation), then there is no need to close it until user is ready to completely exit from using their main software program.

Declaration

```
void ad160_SetClockRate(BYTE CLK_SEL); // 0-3 ==> 60,30,15,7.5 MHz
```

Parameters

CLK_SEL: 0 <= CLK_SEL <= 3 where 0, 1, 2, 3 ==> 60, 30, 15, 7.5 MHz

Return Value

None.

Example

```
ad160_SetClockRate(1); // Set clock rate to 60 Mhz.
```

4.3.2.4 ad160_Set_Memory_Size()**Description**

Sets the exact size of memory that the card will use while digitizing.

Declaration

```
void ad160_Set_Memory_Size(BYTE MemSize);
```

Parameters

MemSize: 4 <= MemSize <= 23

```
/*
if (MemSize == 4) Memory_Size = 16;
if (MemSize == 5) Memory_Size = 32;
if (MemSize == 6) Memory_Size = 64;
if (MemSize == 7) Memory_Size = 128;
if (MemSize == 8) Memory_Size = 256;
if (MemSize == 9) Memory_Size = 512;
if (MemSize == 10) Memory_Size = 1024;
if (MemSize == 11) Memory_Size = 2048;
if (MemSize == 12) Memory_Size = 4096;
if (MemSize == 13) Memory_Size = 8192;
if (MemSize == 14) Memory_Size = 16384;
```

```
    if (MemSize == 15) Memory_Size = 32768;
    if (MemSize == 16) Memory_Size = 65536;
    if (MemSize == 17) Memory_Size = 131072;
    if (MemSize == 18) Memory_Size = 262144;
    if (MemSize == 19) Memory_Size = 524288;
    if (MemSize == 20) Memory_Size = 1048576;
    if (MemSize == 21) Memory_Size = 2097152;
    if (MemSize == 22) Memory_Size = 4194304;
    if (MemSize == 23) Memory_Size = 8388608;
*/
```

Return Value

None.

Example

```
ad160_Set_Memory_Size(23);    // Sets memory size to 8M
```

4.3.2.5 SetTrigPol()**Description**

Sets the edge polarity of the external trigger.

Declaration

```
void SetTrigPol(BYTE TrigPol);
```

Parameters

TrigPol:

- 0: Positive Edge Triggered.
- 1: Negative Edge Triggered.

Return Value

None.

Example

```
SetTrigPol(0);    // Set trigger to Positive Edge Triggered mode.
```

4.3.2.6 ad160_Digitize()**Description**

Sets the segment size. If NumSegments (see "ad160_SetNumSegments") is set to 1, you can use all segment sizes listed below. If using multiple segments (more than 1 segment), then you can only use SegSize: 0 <= SegSize <= 5.

Declaration

```
void ad160_Digitize(BYTE UseExtTrig);
```

Parameters

UseExtTrig:

- 0: Uses automatic internal trigger mode.
- 1: Waits for external trigger to occur before digitizing.

Return Value

None.

Example

```
ad160_Digitize(0);    // Immediately starts digitizing
```

4.3.2.7 ad160_DIG_DONE()**Description**

Returns a “1” value if digitizing is complete, otherwise a “0”. Used as a polling device instead of interrupt for delaying until digitizing is complete before transferring data to system memory.

Declaration

```
BYTE ad160_DIG_DONE(void);
```

Parameters

None.

Return Value

BrdNum: 1 <= CardNum <= 4
 0: Digitizing NOT DONE yet
 1: Digitizing complete

Example

```
i = 0;
while (!ad160_DIG_DONE()) {
    i = i+1;
    if (i > 1000000) break;    // Delay until digitizing is done.
```

4.3.2.8 ad160_Transfer_Data()**Description**

Called after “ad160_DIG_DONE” has come back true to transfer data from card to system memory. The user can call any number of samples up to that previously set by “ad160_Set_Memory_Size()”. This function called be called any number of times as long as the total number does not exceed that set by “ad160_Set_Memory_Size()”.

Declaration

```
BYTE ad160_Transfer_Data(PVOID UserArray, DWORD NumSamples);
```

Parameters

UserArray: Pointer to one dimensional array defined by user.

NumSamples: Any number of samples up to Memory Size previously set.

Return Value

0: Everything is ok.
1: Digitizing did not complete

Example

```
ad160_Transfer_Data(UserArray, 1024);    // Moves 1024 samples to UserArray.
```

4.3.2.9 ad160_PowerDown()

Description

Places AD160 in the lowest possible power state. AD160 must be powered up for TBD microseconds before digitizing.

Declaration

```
void ad160_PowerDown(BYTE PwrDwn);
```

Parameters

PwrDwn:

- 0: Board is Powered "ON" and ready for digitizing
- 1: Forces board into a Powered "OFF" state. Cannot digitize in this state.

Return Value

None.

Example

```
ad160_PowerDown(1); // Puts AD160 card into a low power state.
```

4.4 Programming Examples

4.4.1 DOS 16-BIT C Example

```
/*                                     */
/* AD160 Sample File - Calls "ad160drv.c" */
/* ===== */
/*                                     */
/* 16-Bit DOS Application (tested on MSDOS 6.22) */
/*                                     */
/* Filename: ad160_example.c */
/*                                     */
/* Web site: http://www.chase2000.com */
/* Email: support@chase2000.com */
/*                                     */
/* (c) Chase Scientific 2006 */
/*                                     */

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "ad160drv.c" /* Actual driver routines for AD160 */

typedef WORD *Array; /* One Dimensional Array, 16-bit unsigned */
Array dataArray;

/*****

main ()
{
/*-----*/
```

```

/* 1) INITIALIZE VARIABLES,  ALLOCATE MEMORY                                     */
/*-----*/

    unsigned long i;
    FILE *fout1;

/* Allocate memory for dynamic variable DataArray. See ad160drv.h */
/* for declaration of DataArray.                                     */
    if ((DataArray = (Array) calloc(1024,sizeof(WORD *))) == NULL)
    {
        printf( "Not Enough Memory\n" ) ;
        exit(1) ;
    }

    fout1 = fopen("data1.dat", "w") ;

/*-----*/
/* 2)  SETUP CARD                                                         */
/*-----*/

    ad160_SetBasePortAddr(0x240);
        printf("Set BasePortAddr. \n") ;
        printf("CTRL_PORT(read) = %x\n\n", inportb(CTRL_PORT) ) ;

    ad160_Reset();
        /* Resets Key Control Bits */
        printf("Reset. \n") ;
        printf("CTRL_PORT(read) = %x\n\n", inportb(CTRL_PORT) ) ;

    ad160_PowerDown(0);
        /* Power Up Card */
        printf("De-Activate PWR_DWN. \n") ;
        printf("CTRL_PORT(read) = %x\n\n", inportb(CTRL_PORT) ) ;

    ad160_SetClockRate(1);
        /* Set Clock to 60 MHz */
        printf("SetClockRate to 60 MHz. \n") ;
        printf("CTRL_PORT(read) = %x\n\n", inportb(CTRL_PORT) ) ;

    ad160_Set_Memory_Size(10);
        /* Set Memory Size to 1024 Samples */
        printf("Set Memory Size to 1024 Samples. \n") ;
        printf("SEG_PORT(read) = %x\n\n", inportb(SEG_PORT) ) ;

        printf("Set Trigger Polarity. \n") ;

    SetTrigPol(0);
        printf("CTRL_PORT(read) = %x\n\n", inportb(CTRL_PORT) ) ;

/*-----*/
/* 3)  DIGITIZE / WAIT FOR DONE                                           */
/*-----*/

        printf("Start Digitize. \n") ;

    ad160_Digitize(0);
        printf("CTRL_PORT(read) = %x\n\n", inportb(CTRL_PORT) ) ;

        printf("Waiting for PostTriggering to Complete ... \n") ;

    i = 0;
    while (!ad160_DIG_DONE()) {
        i = i+1;
        if (i > 10000) break;
    }

        printf("DIG_DONE = 1! \n") ;
        printf("Polling Iterations = %d\n\n",i);

/*-----*/
/* 4)  TRANSFER DATA FROM AD160 SRAM TO "DataArray"                     */
/*-----*/

        printf("Transfer Data from Board to System Memory. \n") ;
    ad160_Transfer_Data(DataArray,Memory_Size);
    /* Manually transfers data from CH1 buffer */
    /*
        printf("CTRL_PORT(read) = %x\n\n", inportb(CTRL_PORT) ) ; */

    ad160_Reset();
        /* Resets Key Control Bits */
        printf("Reset. \n") ;
        printf("CTRL_PORT(read) = %x\n\n", inportb(CTRL_PORT) ) ;

```

```

/*-----*/
/* 5)  WRITE CH1 DATA TO FILE "Data1.dat" */
/*-----*/
        printf("Writing CH1 Data from Memory to Files Data1.dat \n") ;
        for ( i = 0; i < (Memory_Size-1); i++) /* Memory_Size; i++ ) */
        {
/*          printf( "%d\n", (DataArray[i] & 0xFFFF) ) ; */
            fprintf( fout1, "%d\n", (DataArray[i] & 0xFFFF) );
        }

        printf("Writing Complete. \n\n") ;

        printf("CTRL_PORT(read) = %x\n", inportb(CTRL_PORT) ) ;
        fprintf( fout1, "%d\n", inportb(CTRL_PORT) );

/*-----*/
/* 6)  FREE "DataArray" Memory, CLOSE FILE fout1 ("Data1.dat") */
/*-----*/

        free(DataArray);
        fclose( fout1 ) ;

        return (0);
}

```

4.4.2 LINUX C/C++ Example

```

//-----
// Filename: ad160_test.c
//-----
//
// 6-10-2006          Debug on Fedora 5 (Linux Version 2.6.15)
//                   - Downloaded 3/20/06
//
//-----
// Web site: http://www.chase2000.com
// Email:      support@chase2000.com
//
// (c) Chase Scientific 2006
//
//-----
//-----

#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <math.h>
#include <asm/io.h> //<sys/io.h> // Supersedes "asm/io.h"

#include "ad160drv_o.H" // Actual driver routines for AD160

/*****

int main() //(int argc, char *argv[])
{
//-----
// 1) INITIALIZE VARIABLES
//-----

typedef WORD *Array; // One Dimensional Array, 16-bit unsigned
Array DataArray;

```

```

unsigned long MemorySize;
int i;

FILE *fout1;
fout1 = fopen("data1.txt", "w") ;

//-----
// 2)  SETUP CARD
//-----

ad160_SetBasePortAddr(0x240);
printf("Set BasePortAddr. \n") ;
printf("CTRL_PORT(read) = %x\n\n", inb(0x240) ) ;

ad160_Reset();           // Resets Key Control Bits
printf("Reset. \n") ;
printf("CTRL_PORT(read) = %x\n\n", inb(0x240) ) ;

// ad160_PowerDown(1);    // Power Down Card Test (activate PWRDWN)
// printf("Activate PWR_DWN (high). \n") ;
// printf("CTRL_PORT(read) = %x\n\n", inb(0x240) ) ;
// return (0);           // and return

ad160_PowerDown(0);      // Power Up Card (deactivate PWRDWN)
printf("De-Activate PWR_DWN. \n") ;
printf("CTRL_PORT(read) = %x\n\n", inb(0x240) ) ;

ad160_SetClockRate(1);   // Set Clock to 7.5 MHz  1=60, 2=30, 3=15, 4=7.5
printf("SetClockRate to 7.5MHz. \n") ;
printf("CTRL_PORT(read) = %x\n\n", inb(0x240) ) ;

ad160_Set_Memory_Size(17); // Set Memory Size to 128K
MemorySize = 131072;
// ad160_Set_Memory_Size(23); // Set Memory Size to 128K
// MemorySize = 8388608;
// ad160_Set_Memory_Size(10); // Set Memory Size to 128K
// MemorySize = 1024;
printf("Set Memory Size to 8M Samples. \n") ;
printf("SEG_PORT(read) = %x\n\n", inb(0x241) ) ;

//-----
// 3)  ALLOCATE MEMORY FOR "DataArray"
//-----

if ((DataArray = (Array) calloc(MemorySize,sizeof(WORD *))) == NULL)
{
    printf( "Not Enough Memory\n" ) ;
    exit(1) ;
}

//-----
// 4)  DIGITIZE / WAIT FOR DONE
//-----

SetTrigPol(0);
printf("Set Trigger Polarity. \n") ;
printf("CTRL_PORT(read) = %x\n\n", inb(0x240) ) ;

ad160_Digitize(0);
printf("Start Digitize. \n") ;
printf("CTRL_PORT(read) = %x\n\n", inb(0x240) ) ;

printf("Waiting for PostTriggering to Complete ... \n") ;

i = 0;
while (!ad160_DIG_DONE()) {
    i = i+1;
    if (i > 1000000) break;
}

printf("DIG_DONE = 1! \n") ;
printf("Polling Iterations = %d\n\n",i);

```

```

//-----
// 5)  TRANSFER DATA FROM AD160 SRAM TO "DataArray"
//-----

    ad160_Transfer_Data(DataArray,MemorySize);
                                printf("Transfer Data from Board to System Memory. \n") ;
                                printf("CTRL_PORT(read) = %x\n\n", inb(0x240) ) ;

//  ad160_Reset();                // Resets Key Control Bits
//                                printf("Reset. \n") ;
//                                printf("CTRL_PORT(read) = %x\n\n", inb(0x240) ) ;

//-----
// 6)  WRITE CH1 DATA TO FILE "Data1.txt"
//-----

                                printf("Writing CH1 Data from Memory to Files Data1.txt \n") ;
    for ( i = 0; i < (MemorySize); i++)
    {
//                                printf( "%d\n", (DataArray[i] & 0xFFFF) ) ;
        fprintf( fout1, "%d\n", (DataArray[i] & 0xFFFF) );
    }
                                printf("Writing Complete. \n\n") ;

                                printf("CTRL_PORT(read) = %x\n", inb(0x240) ) ;
//  fprintf( fout1, "%d\n", inb(CTRL_PORT) );

//-----
// 7)  FREE "DataArray" Memory, CLOSE FILE fout1 ("Data1.txt")
//-----

    free(DataArray);
    fclose( fout1 ) ;

    return (0);
}

```

5 MISCELLANEOUS

5.1 Calibration

The ad160 has no user feature to calibrate for input and output gain accuracy or offsets. The gains and offsets are calibrated at the factory to be within 3% at 25°C and nominal voltages with inputs terminated by 50 ohms source.

5.2 Maintenance

No maintenance is required. However, a yearly calibration is recommended if the user desires to maintain the ad160 modules' specified accuracy. Call factory for maintenance and/or extended warranty information.

Trademarks:

MS-DOS, Windows 3.1, Windows 95, Windows NT, Windows 2000, and Windows XP are registered trademarks of Microsoft Corporation.